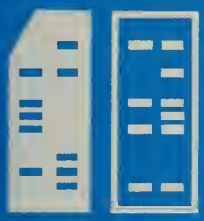
REPORT NO. UIUCDCS-R72-509

COST EFFECTIVE PRIORITY ASSIGNMENT IN NETWORK COMPUTERS

by

E. K. Bowdon, Sr. and W. J. Barr

March 28, 1972



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

REPORT NO. 509

COST EFFECTIVE PRIORITY ASSIGNMENT IN NETWORK COMPUTERS

by

E. K. Bowdon, Sr. and W. J. Barr

Department of Computer Science
University of Illinois
Urbana, Illinois

(This research was supported in part by the National Science Foundation under Grant No. NSF GJ 28289.)

ABSTRACT

With the advent of network computers, a new area of computer systems analysis has evolved. Unfortunately, most of the work which has been done to date merely extends the previously existing theory of communications. While this work has been very fruitful and produced important results, our analysis is predicated on the assumption that a geographically distributed network computer is, in reality, quite different from telephone networks and individual computing centers.

In this paper, we focus our attention on the probable goals of the networks and define a measure of cost effectiveness. Then using this measure we develop a priority assignment technique for the individual centers that comprise the network. We conclude by expanding the measure of cost effectiveness to determine load leveling rules for the entire network.

1. INTRODUCTION

Previously, the study of network computers has been focused on the analysis of communication costs, optimal message routing, and the construction of a communications network connecting geographically distributed computing centers. While these problems are far from being completely solved, enough progress has been made to allow the construction of reasonably efficient network computers. One problem which has not been solved, however, is making such networks economically viable. The solution of this problem is the object of our analysis.

Our basic assumption is that economic viability for network computers is predicated on efficient resource sharing. This was, in fact, a major reason for the construction of several networks — to create the capability of using someone else's special purpose machine or unique process without having to physically transport the work. This type of resource sharing is easily implemented and considerable work has been done towards this goal. There is, however, another aspect of resource sharing which has not been studied thoroughly: load-leveling. By load-leveling we mean the transfer of tasks between computing centers for the purpose of improving the throughput of the network or other criteria. We contend that the analysis and implementation of user-oriented load-leveling is the key to developing economically self-supporting network computers.

2. A SCENARIO OF COST EFFECTIVENESS

Until recently, efforts to measure computer efficiency have centered on the measurement of resource (including processor) idle time. A major problem with this philosophy is that it assumes that all tasks are of roughly equal value to the user and hence the operation of the system.

As an alternative to the methods used in the past, we propose a priority assignment technique designed to represent the worth of tasks in the system. We present the hypothesis that tasks requiring equivalent use of resources are not necessarily of equivalent worth to the user with respect to time. We would allow the option for the user to specify a "deadline" after which the value of his task would decrease, at a rate which he can specify, to a system determined minimum. With this in mind, we propose a measure of cost effectiveness with which we can evaluate the performance of a network with an arbitrary number of interconnected systems, as well as each system individually.

We define our measure of cost effectiveness γ, as follows:

$$\gamma = \left(\frac{L_q}{M}\right)^{\alpha} \left(1 - \frac{M - L_q}{\sum_{i=0}^{R-1} \beta(i)}\right)$$

where

L_q is the number of tasks in the queue,

M is the maximum length queue,

R is the number of priority classes,

α is a measure (system-determined) of the "dedicatedness" of the CPU to the processing of tasks in the queue,

and $$\beta(i) = (R-i) \sum_{j=1}^{n} (g(j)/f(j))$$

where

$g(j)$ is the reward for completing task j (a user specified function of time),

and $f(j)$ is the cost (system determined) to complete task j.

The term $(\frac{L_q}{M})^{\alpha}$ is a measure of the relevance of the queue to processing activities. Similarily, we can look at $\beta(i)$ as a measure of resource utilization. Note that $\beta(i)$ indicates a ratio of reward to cost for a given priority class and is sensitive to the needs of the user and the requirements imposed on the installation. It is user sensitive because the user specifies the reward and is installation sensitive because the cost for processing a task is determined by the system. The measure of CPU dedicatedness (α), on the other hand, is an entirely installation sensitive parameter.

The first problem which becomes apparent is that which arises if $\sum_{i=0}^{R-1} \beta(i) = 0$. This occurs only in the situation where there is exactly one priority class (i.e., the non-priority case). We will finesse away this problem by defining

$$\frac{M-L_q}{\sum_{i=0}^{R-1} \beta(i)} = 0$$

for this case. Intuitively, this is obvious, since the smaller this term gets, the more efficiently (in terms of reward) a system is using its resources.

Furthermore, in the absence of priorities, the order in which tasks are executed is fixed, so this term becomes irrelevant to our measure of cost effectiveness. Thus, for the non-priority case, we have

$$\gamma = \left(\frac{L_q}{M}\right)^{\alpha}$$

which is simply the measure of the relevance of the queue to processing activities. This is precisely what we want if we are going to consider only load-leveling in non-priority systems. However, we are interested in the more general case in which we can assign priorities.

An estimate of the cost to complete task j, f(j) is readily determined from the user-supplied parameters requesting resources. Frequently these estimated parameters are used as upper limits in the resource allocation and the operating system will not allow the program to exceed them. As a result, the estimates tend to be high. On the other hand, lower priorities are usually assigned to tasks requiring a large amount of resources. So the net effect is, that the user's parameters reflect his best estimate and we may be reasonably confident that they truly reflect his needs.

At the University of Illinois computing center, for example, as of July 26, 1971, program charges are estimated by the following formula:

$$\text{cents} = a(X + Y)(bZ + c) + d$$

where X = CPU time in centiseconds,

Y = number of I/Ø requests,

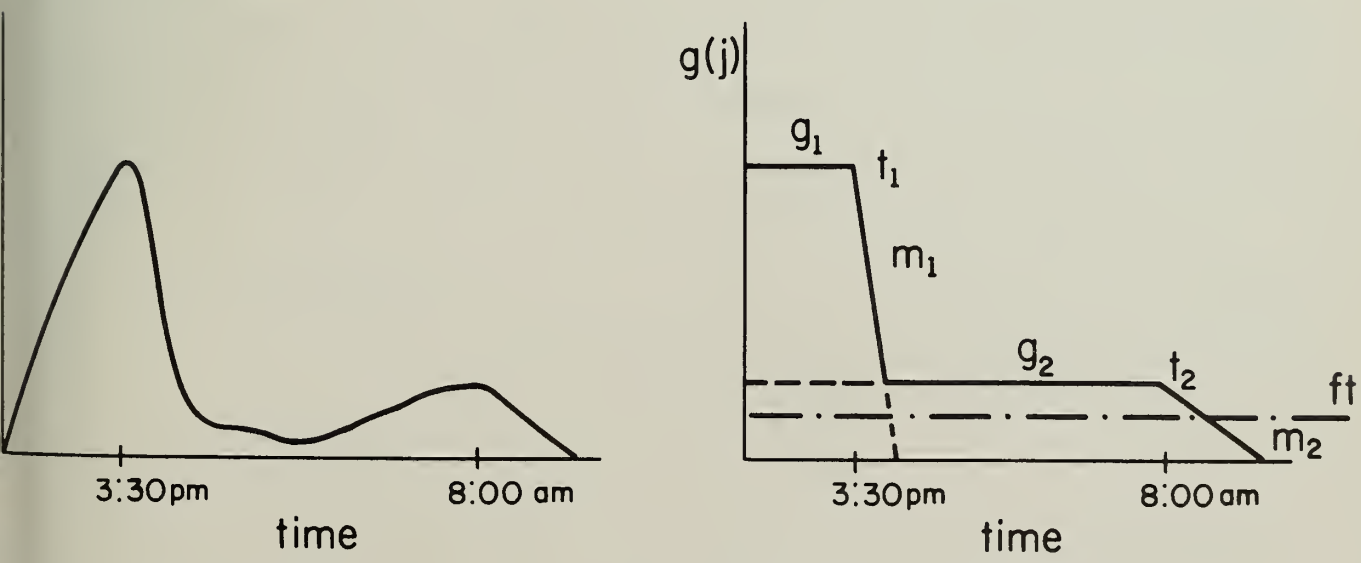
Z = core size in kilobytes,

a, b, c are weighting factors currently having the values 0.04, 0.0045, and 0.5, respectively,

and d is an extra charge factor including $1.00 cover charge plus any special resources used (tape/disk storage, card read, cards punched, plotter, etc.).

The main significance of the reward function g(j) specified by the user is that it allows us to determine a deadline or deadlines for the task. Typically we might expect g(j) to be a polynomial in t, where t is the time in the system. For example, the following thoughts might run through the user's head: "Let's see, its 10:00 a.m. now and I don't really need immediate results since I have other things to do. However, I do need the output before the 3:00 p.m. meeting. Therefore, I will make 2:30 p.m. a primary deadline. If it isn't done before the meeting, I can't use the results before tomorrow morning, so I will make 8:00 a.m. a secondary deadline. If it isn't done by then I can't use the results, so after 8:00 a.m. I don't care."

The function g(j) this user is thinking about would probably look something like Figure 1a. Now, this type of function poses a problem in that it is difficult for the user to specify accurately and would require an appreciable amount of overhead to remember and compute. Notice, however, that even if turnaround time is immediate, the profit oriented installation manager would put the completed task on a shelf (presumably an inexpensive storage device)



(a) Ideal function.

(b) Approximate function.

Figure 1. Example of a User's Reward Function.

and not give it to the user until just before the deadline-thus collecting the maximum reward. As a result, there is little reason to specifying anything more than the deadlines, the rewards associated with meeting the deadlines, and the rate of decrease of the reward between deadlines, if any. Applying this reasoning to Figure 1a we obtain Figure 1b. Note that this function is completely specified with only six parameters (deadlines t_1, t_2; rewards g_1, g_2; and rates of decrease m_1, m_2).

In general, we may assume that g(j) is a monotonically non-increasing, piecewise linear, reward function consisting of n distinct sets of deadlines, rewards, and rates of decrease. Thus we can simply specify g(j) with 3n parameters.

Note that, in effect, the user specifies an abort time when the g(j) he specifies becomes less than f(j). If the installation happens to provide a "lower cost" service, $\tilde{f}(j)$ and if $g(j) > \tilde{f}(j)$, this task would be processed, but only when all the tasks with higher g(j) had been processed.

Now, what we are really interested in, is not so much an absolute reward, but a ratio of reward to cost. Since f(j) is, at best, only an estimate of cost, we cannot reasonably require a user to specify an absolute reward. A more equitable arrangement would be to specify the rewards in terms of a ratio g(j)/f(j) associated with each deadline. This ratio is more indicative of the relative worth of a task, both to the system and to the user, since it indicates the return on an investment.

2. PRIORITY ASSIGNMENT

Let us now turn our attention to the development of a priority assignment scheme which utilizes the reward/cost ratios described in the previous section. We begin by quantizing the continium of reward/cost ratios into R distinct intervals. Each of these intervals is then assigned to one of R priority classes 0, 1, 2, ..., R-1 with priority 0 being reserved for tasks with highest reward/cost ratios and priority R-1 for tasks with reward/cost ratios of unity or less. A task entering the system will be assigned a priority according to its associated reward/cost ratio.

We want to guarantee, if possible, that all priority 0 tasks will meet their deadlines. Furthermore, if all priority 0 tasks can meet their deadlines, we want to guarantee, if possible, that all priority 1 tasks will meet their deadlines and, in general, if all priority k tasks can meet their deadlines, we want to guarantee that as many priority class k+1 tasks as possible will meet their deadlines. To facilitate the priority assignment, we introduce the following notation:

For priority k, let T_i denote the i^{th} task. Then we assume for each T_i that we receive the following information vector:

$$(T_i, f/g, d_i, \tau_i, s_i)$$

where

T_i is an identifier,

f/g is the reward/cost ratio associated with meeting the task's deadline,

d_i is the task's deadline associated with f/g,

τ_i is the maximum processing time for the task,

and $s_i = d_i - \tau_i$, is the latest time at which the task may start processing and still be assured of meeting its deadline.

Now since each task has an associated deadline and maximum processing time, we can use the resulting latest start time as the basis for assigning positions to tasks within a priority class. A last come, first served rule will be used to break ties. Additionally, we will use a compacting scheme to ensure that as many tasks as possible start processing before their latest start times. More formally our algorithm may be stated as follows:

Priority Assignment Algorithm

First, we assign a new task a priority based on its f/g ratio, say priority k. Then within class k, its position is determined as follows:

1. Beginning with the last priority k task (that is, the task with latest deadline) search forward until two tasks, T_{j-1} and T_j, are found such that $d_{j-1} < d_i \leq d_j$. Insert T_i between the two tasks, assign it a start time $s_i = d_i - \tau_i$, and renumber the tasks behind T_j accordingly (i.e., T_i becomes T_j, T_j becomes T_{j+1}, etc.).
2. Now, if $s_{j-1} + \tau_{j-1} \leq s_j \leq s_{j+1} - \tau_j$ there is sufficient float time between T_{j-1} and T_{j+1} for T_j to be processed on time and the priority assignment is complete. However, if either $s_{j-1} + \tau_{j-1} > s_j$ or $s_j + \tau_j > s_{j+1}$, a deadline might be missed; so we proceed with Step 3.
3. Compacting Scheme. Let f_j denote the float time between any two tasks T_{j-1} and T_j, where f_j is defined:

$$f_j = s_j - (s_{j-1} + \tau_{j-1})$$

Then, F_j, the total float time preceding T_j, is given by:

$$F_j = \sum_{k=1}^{j} f_j = s_j - t + \sum_{k=1}^{j} \tau_j.$$

where t is the current time.

Now, starting with task T_j, if $s_j + \tau_j > s_{j+1}$ and $F_j \geq \tau_j$, we assign a new starting time to T_j given by:

$$s_j = s_{j+1} - \tau_j.$$

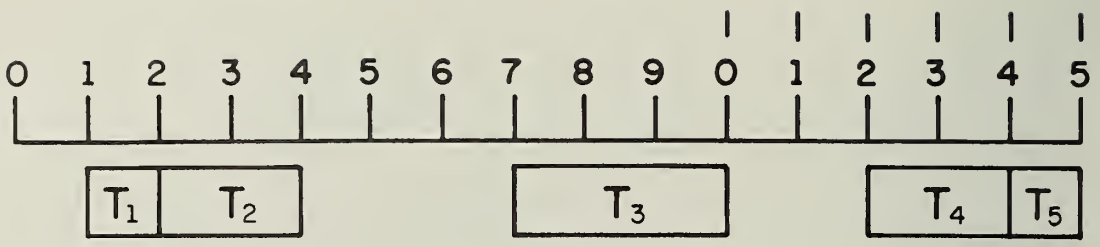
and we continue with T_{j-1}, T_{j-2}, etc. until we encounter a task T_k, $k \neq j$, such that $s_k \leq s_{k+1} - \tau_k$. (Note that T_{j+1} and all its predecessors are guaranteed to meet their deadlines.)

4. However, if $s_j + \tau_j > s_{j+1}$ but $F_j < \tau_j$, we do not assign a new start time to T_j. Instead we leave the start time at its latest critical value, even though it may not start processing at that time. We observe that many tasks may not require all of the processing time specified by their maxima, and as a result sufficient float time may be created later to enable the task, T_j, to meet its deadline.

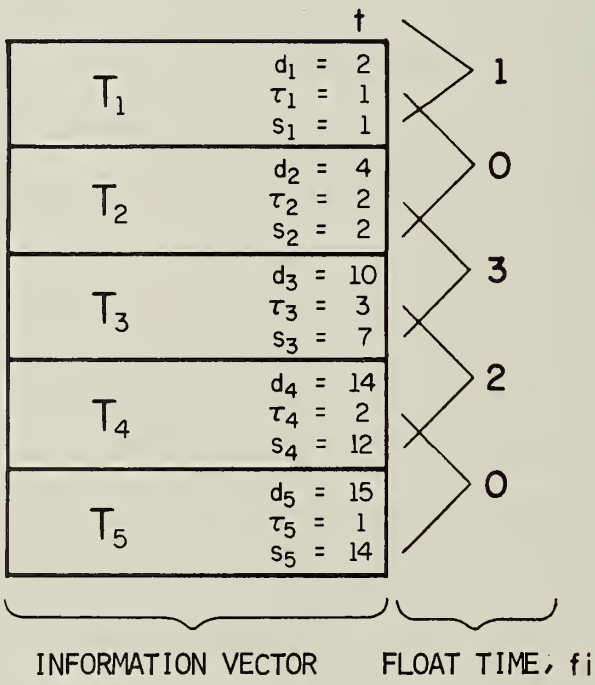
Examples.

Several examples will now be given to illustrate the efficacy of the algorithm. Suppose we have determined that a task, T_i, should have priority k and that at time t = 0, the state of priority class k is that shown in Figure 2. (Note that since all priority class k tasks have similar f/g, we need not show these ratios.) Notice that forming the float time column is analogous to forming a forward difference table. In each of the following examples we assume that Figure 2 is the initial state of priority class k.

i) Suppose the information vector (with f/g omitted) for T_i is $(T_i,6,1,5)$. Beginning with T_5, we observe that $d_2 < d_i \leq d_3 < d_4 < d_5$. So we insert T_i between T_2 and T_3 and renumber the tasks accordingly.



A) SCHEDULE OF TASKS.



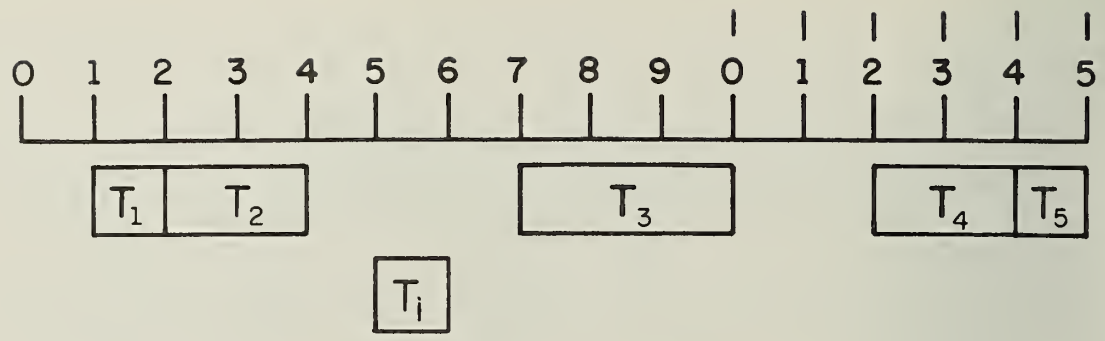
B) INFORMATION VECTORS.

FIGURE 2 – STATE OF PRIORITY CLASS K AT TIME $t = 0$

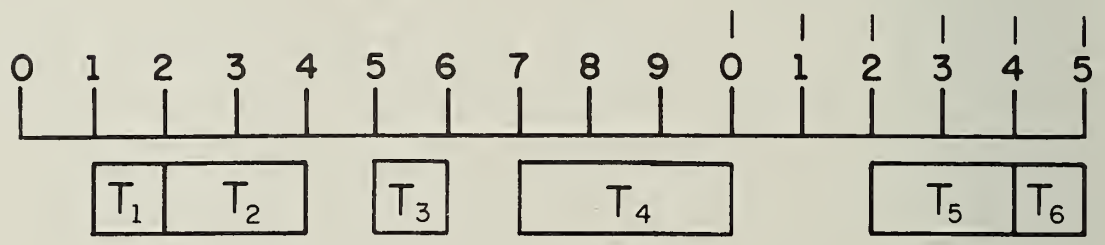
Now $s_2 + \tau_2 \leq s_3 \leq s_4 - \tau_3$ since $2 + 2 \leq 5 \leq 7 - 1$, so the priority assignment is complete and all tasks are guaranteed to meet their deadlines. The resulting state of priority class k is shown in Figure 3.

ii) Suppose instead that the information vector for T_i is $(T_i,9,2,7)$. We find that $d_2 < d_i \leq d_3 < d_4 < d_5$, so we insert T_i between T_2 and T_3 and renumber the tasks accordingly. However, $s_3 + \tau_3 > s_4$ since $7 + 2 > 7$ and a deadline could be missed. But $F_3 \geq \tau_3$ since $4 \geq 3$, so we assign a new start time to T_3: $s_3 = s_4 - \tau_3 = 7 - 2 = 5$. Now $s_2 \leq s_3 - \tau_2$ since $2 \leq 5 - 2$, so the priority assignment is complete and all tasks are guaranteed to meet their deadlines. The resulting state of priority class k is shown in Figure 4. (Note the effect of the last in first out rule for breaking ties on start times.)

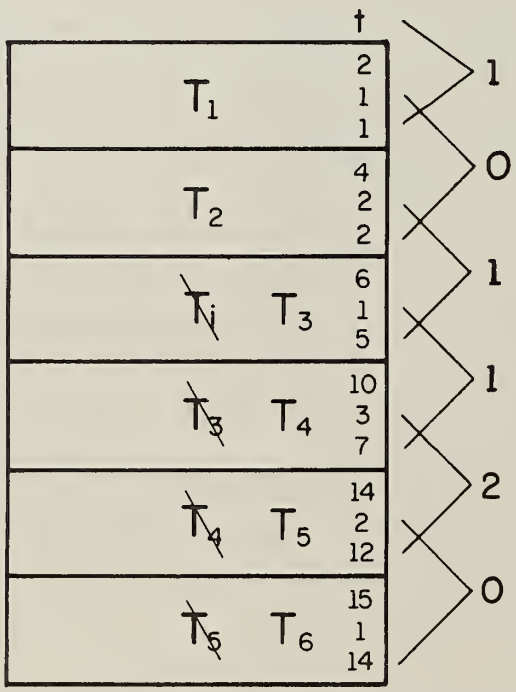
iii) Next suppose the information vector for T_i is $(T_i,9,4,5)$. We find that $d_2 < d_i \leq d_3 < d_4 < d_5$, so we insert T_i between T_2 and T_3 and renumber the tasks accordingly. However, $s_3 + \tau_3 > s_4$ since $5 + 4 > 7$, and a deadline could be missed. But $F_3 \geq \tau_3$ since $4 \geq 4$, so we assign a new start time to T_3: $s_3 = s_4 - \tau_3 = 7 - 4 = 3$. Next $s_2 + \tau_2 > s_3$ since $2 + 2 > 3$, so we assign a new start time to T_2: $s_2 = s_3 - \tau_2 = 3 - 2 = 1$. Now $s_1 + \tau_1 > s_2$ since $1 + 1 > 1$, so we assign a new start time to T_1: $s_1 = s_2 - \tau_1 = 0$. The priority assignment is complete and all tasks are guaranteed to meet their deadlines. The resulting state of priority class k is shown in Figure 5.



A) SCHEDULE OF TASKS BEFORE ASSIGNMENT.

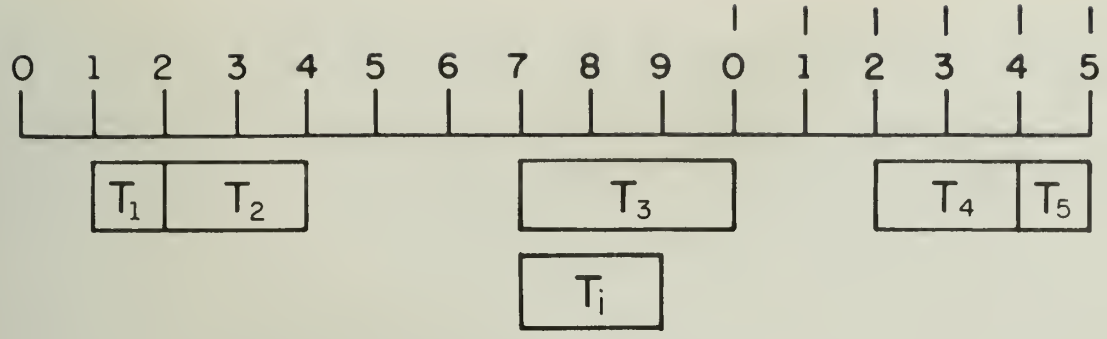


B) SCHEDULE OF TASKS AFTER ASSIGNMENT.

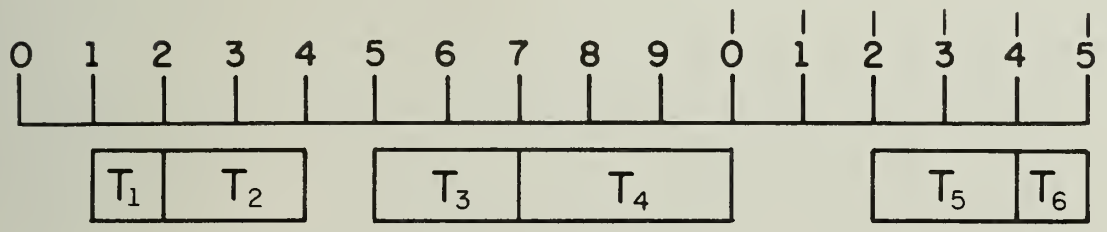


C) INFORMATION VECTORS AFTER ASSIGNMENT.

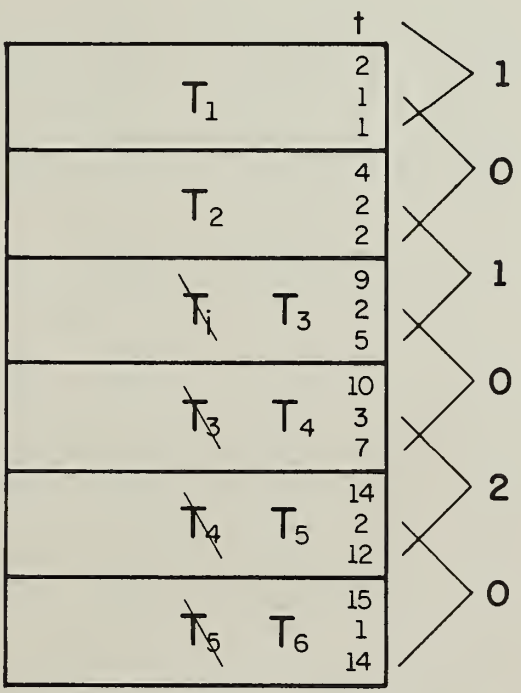
FIG.3 - RESULTS OF PRIORITY ASSIGNMENTS FOR EXAMPLE (i)



A) SCHEDULE OF TASKS BEFORE ASSIGNMENT.

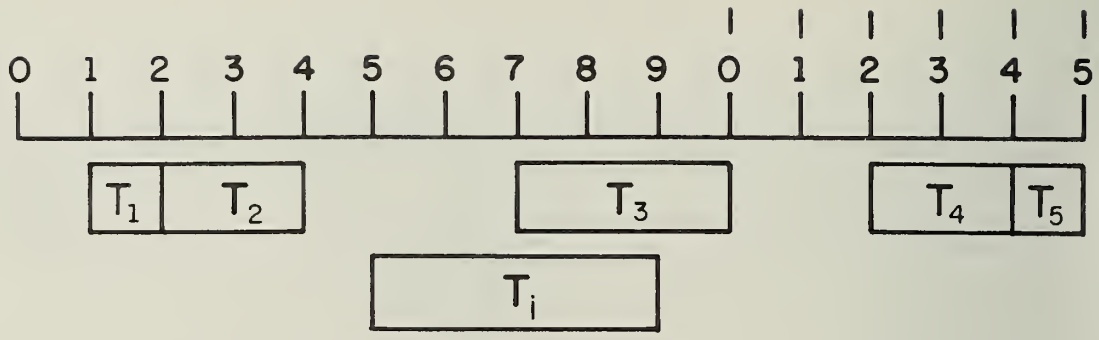


B) SCHEDULE OF TASKS AFTER ASSIGNMENT.

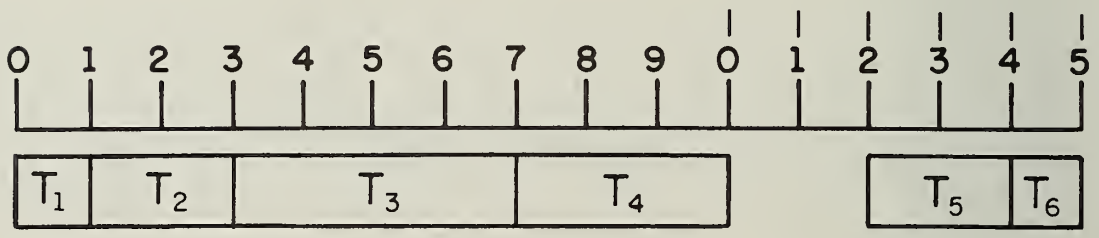


C) INFORMATION VECTORS AFTER ASSIGNMENT.

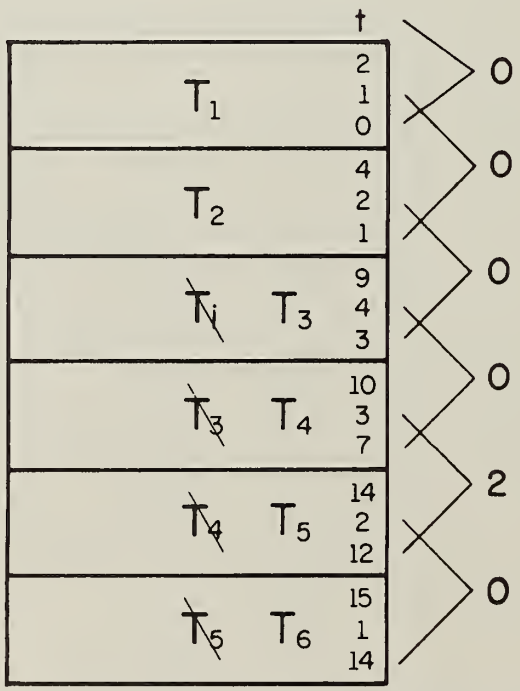
FIG. 4 - RESULTS OF PRIORITY ASSIGNMENTS FOR EXAMPLE (ii)



A) SCHEDULE OF TASKS BEFORE ASSIGNMENT.



B) SCHEDULE OF TASKS AFTER ASSIGNMENT.



C) INFORMATION VECTORS AFTER ASSIGNMENT.

FIG.5 - RESULTS OF PRIORITY ASSIGNMENTS FOR EXAMPLE (iii)

iv) As a final example, suppose that the information vector for T_i is $(T_i,9,5,4)$. As before, we find that $d_2 < d_i \leq d_3 < d_4 < d_5$, so we insert T_i between T_2 and T_3 and renumber the tasks accordingly. However, $s_3 + \tau_3 > s_4$ since $4 + 5 > 7$, and a deadline could be missed. Furthermore, $F_3 < \tau_3$ since $4 < 5$, and the compacting scheme will not help us. Instead we leave the start times at their latest critical values and hope that sufficient float time is created later to enable the tasks to meet their deadlines. The results of this assignment are shown in Figure 6. Note that T_5 is the task which is in danger of missing its deadline.

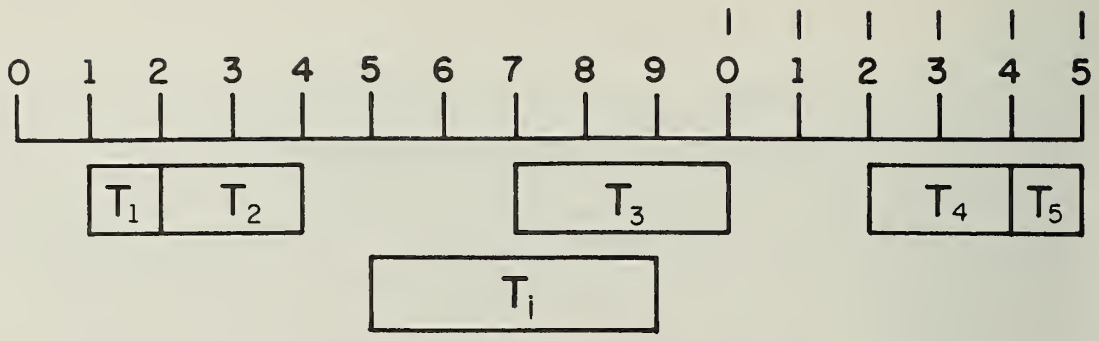
The last example brings up the problem of what to do with a task whose deadline is missed. We simply treat it as though it had just entered the system using the next specified deadline as the current deadline. If no further deadlines are specified, the task is assigned priority R-1 and will be processed accordingly.

When a processor finishes executing a task the following scheduling algorithm is used to determine which task is to be processed next. Generally, the algorithm takes the highest priority task in the queue that is closest to its latest starting time.

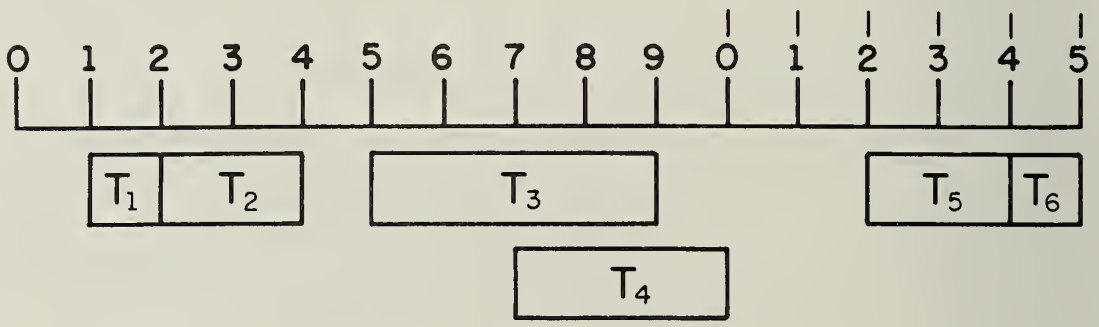
Scheduling Algorithm

Beginning with k=0.

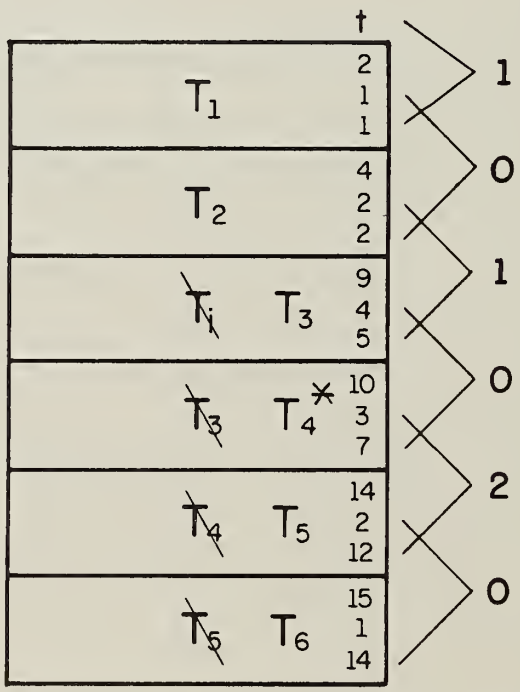
1. We examine the float time, f_1, for the first task in priority class k. Then for ℓ = k+1:



A) SCHEDULE OF TASKS BEFORE ASSIGNMENT.



B) SCHEDULE OF TASKS AFTER ASSIGNMENT.



C) INFORMATION VECTORS AFTER ASSIGNMENT.

FIG.6 - RESULTS OF PRIORITY ASSIGNMENTS FOR EXAMPLE (iv)

2. If f_1 of priority class $k \leq \tau_1$ for the first task of priority class ℓ, we set $k=\ell$ and continue with Step 1. Otherwise we continue with Step 3.

3. Set $\ell = \ell+1$ and continue with Step 2 until all priority classes have been considered. Then continue with Step 4.

4. Assign the first task, T_1, in priority k to the available processor.

In the foregoing we have tacitly assumed that each task enters the system sufficiently before its deadline to allow processing. The two algorithms taken together facilitate meeting the deadlines, where possible, of the higher priority tasks. Those tasks which do not meet their deadlines will tend to be uniformly late.

3. LOAD LEVELING IN A NETWORK OF CENTERS

Thus far we have been concerned only with cost effectiveness in a single center. Next, let us consider the more general problem of load leveling within a network of centers. Each center may contain a single computer or a subnet of computers. The topological and physical properties of such networks have been illustrated (1,5,6,7) and will not be discussed here.

We wish to determine a strategy which optimizes the value of work performed by the network computer. That is, to guarantee that every task in each center will be processed, if possible, before its deadline and only those tasks that offer the least reward to the network will miss their deadlines. Implicit in this discussion is the simplifying assumption that any task can be performed in any center. Removal of this restrictive assumption leads to many interesting problems for future research.

We define the measure of cost effectiveness for a network of N centers, γ_N, as follows:

$$\gamma_N = \sum_{i=1}^{N} \omega_i \gamma_i \qquad \text{given that} \qquad \sum_{i=1}^{N} \omega_i = 1$$

where

the ω_i are weighting factors that reflect the relative contribution of the i^{th} center to the overall computational capability of the network,

and γ_i is the measure of cost effectiveness for the i^{th} center.

Note that if a center is a subnet of computers, we could employ this definition to determine the measure of cost effectiveness for the subnet. We also let c_{ij} denote the cost of communication between centers i and j; and t_{ij} as the transmission time between centers i and j.

Ideally, we want the network computer to operate so that all tasks within the network are processed before their deadlines. If a task is in danger of missing its deadline, we want to consider it as a candidate for transmission to another center for processing. The determination of which tasks should be transferred follows the priority assignment (i.e., priority 0 tasks in danger of missing deadlines should be the first to be considered, priority 1 tasks next, etc.).

We note that this scheme may not discover all tasks that are in danger of missing their deadlines. In order to discover all tasks that might be in danger of missing their deadlines, we would require a look ahead scheme to determine the available float time and to fit lower priority tasks into this float time. The value of such a scheme is questionable, however, since we assume some float time is created during processing and additional float time may be created by sending high priority tasks to other centers. Also, the overhead associated with executing the look ahead scheme would further reduce the probable gain of such a scheme.

The determination of which center should be the recipient of a transmitted task can be determined from the measure of cost effectiveness of each center. Recall that the measure indicates the worth of the work to be processed within a center. Thus, a center with a task in danger of missing its deadline will have a larger measure than a center with available float time. Thus, by examining the measures for each center, we can determine the likely recipient of tasks to be transmitted. These centers can in turn, examine their own queues and bid for additional work on the basis of their available float time. This approach has a decided economic advantage over broadcasting the availability of work throughout the network and transmitting the tasks to the first center to respond (4).

Once a recipient center has been determined, we would transmit a given task only if the loss in reward associated with not meeting its deadline is greater than c_{ij}, the cost of transmitting the task between centers and transmitting back the results.

When a task is transmitted to a new center its deadline is diminished by t_{ij}, the time to transmit back the results, thus ensuring the task will reach its destination before its true deadline. Similarly, the reward associated with meeting the task's deadline is diminished by c_{ij}, since this represents a reduction in profit. Then the task's f/g ratio is used to determine a new priority and the task is treated like one originating in that center.

This huristic algorithm provides the desired results that within each center all deadlines are met, if possible, and if any task is in danger of missing its deadline, it is considered for possible transmission to another center which can meet the deadline.

4. SUMMARY

We have introduced a priority assignment technique which, together with the scheduling algorithm, provides a new approach to resource allocation. The most important innovation in this approach is that it allows a computing installation to maximize reward for the use of resources while allowing the user to specify deadlines for his results. The demand by users upon the resources of a computing installation is translated into rewards for the center.

This approach offers advantages to the user and to the computing installation. The user can exercise control over the processing of his task by specifying its reward/cost ratio which, in turn, determines the importance the installation attaches to his requests. The increased flexibility to the user in specifying rewards for meeting deadlines yields increased reward to the center. Thus the computing installation becomes cost effective, since for a given interval of time, the installation can process those tasks which return the maximum reward. A notable point here, is that this system readily lends itself to measurement.

The measure of cost effectiveness is designed to reflect the status of a center using the priority assignment technique. From its definition, the value of the measure depends not only on the presence of tasks in the system but upon the priority of these tasks. Thus the measure reflects the worth of the tasks awaiting execution rather than just the number of tasks. Therefore, the measure can be used both, statically, to record the operation of a center and, dynamically, to determine the probability of available float time. This attribute enables us to predict the worth of the work to be performed in any center in the network and facilitates load-leveling between centers.

Finally, our approach treats a network computer as the economic entity that it should be: a market place in which vendors compete for customers and in which users contend for scarce resources. The development of this approach is a first step in the long road to achieving economic viability in network computers.

REFERENCES

1. Abramson, Norman, The ALØHA System, University of Hawaii Technical Report, January, 1972.

2. Bowdon, E. K., Sr., "Dispatching in Network Computers," Proceedings of the Symposium on Computer-Communications, Networks and Teletraffic, April, 1972.

3. Bowdon, E. K., Sr., and W. J. Barr, "Throughput Optimization in Network Computers," Proceedings of the Fifth International Conference of Systems Sciences, Honolulu, January, 1972.

4. Farber, David J., and Kenneth C. Larson, Supplement to Proposal for Research Submitted to the National Science Foundation on Distributed Computing System, University of California at Irvine, Technical Report, October, 1970.

5. Frank, H., and I. T. Frisch, Communication, Transmission and Transportation Networks, Addison-Wesley, 1971.

6. Kleinrock, L., Communication Nets; Stochastic Flow and Delay, McGraw-Hill, New York, 1964.

7. Syski, R., Introduction to Congestion Theory in Telephone Systems, Oliver and Boyd, Edinburgh, 1960.

IBLIOGRAPHIC DATA HEET	1. Report No. UIUCDCS-R72-509	2.	3. Recipient's Accession No.
Title and Subtitle: Cost Effective Priority Assignment in Network Computers			5. Report Date: March 1972 6.
Author(s): Edward K. Bowdon, Sr. and William J. Barr			8. Performing Organization Rept. No.
Performing Organization Name and Address: Department of Computer Science, University of Illinois, Urbana, Illinois 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. NSF GJ 28289
2. Sponsoring Organization Name and Address: National Science Foundation, Washington, DC			13. Type of Report & Period Covered: Research 14.

5. Supplementary Notes

6. Abstracts

With the advent of network computers, a new area of computer systems analysis as evolved. Unfortunately, most of the work which has been done to date merely xtends the previously existing theory of communications. While this work has been ery fruitful and produced important results, our analysis is predicated on the ssumption that a geographically distributed network computer is, in reality, quite ifferent from telephone networks and individual computing centers.

In this paper, we focus our attention on the probable goals of the networks nd define a measure of cost effectiveness. Then using this measure we develop a riority assignment technique for the individual centers that comprise the network. e conclude by expanding the measure of cost effectiveness to determine load leveling ules for the entire network.

7. Key Words and Document Analysis. 17a. Descriptors

Network Computer, Queueing Theory, System Modeling, Throughput Analysis, Dispatching, Load Leveling

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

Availability Statement	19. Security Class (This Report)	21. No. of Pages
Release unlimited	UNCLASSIFIED	23
	20. Security Class (This Page) UNCLASSIFIED	22. Price